ハイパフォーマンス
コンピューティング 52－4
（1994．7．22）

AP1000+: 並列化コンパイラをサポートするアーキテクチャ

林 憲一　土肥 実久　堀江 健志　小柳 洋一　白木 長武
進藤 達也　今村 信貴　清水 俊幸　石畑 宏明
(株)富士通研究所　並列処理研究センター
E-mail: woods@flab.fujitsu.co.jp

概要

分散メモリ型高並列計算機 AP1000+ は，HPF，VPP Fortran などの言語の仕様に基づき，これらの言語が効率良く実行できるように必要とされる通信機能を検討して開発された．AP1000+ では，並列化コンパイラの利用する通信機能を効率的に実現するために，ノンブロッキング通信，ブロック転送，ストライドデータ転送，グローバル演算，バリア同期などをアーキテクチャでサポートしている．本論文では，これら AP1000+ の機能のうち，ノンブロッキング通信に関連するキューの溢れの影響，またストライドデータ転送の性能への効果について VPP Fortran で書かれた NAS パラレルベンチマーク等を用いて，シミュレーションによって評価する．

AP1000+: Architectural Support for Parallelizing Compiler

Kenichi Hayashi, Tsunehisa Doi, Takeshi Horie, Yoichi Koyanagi,
Osamu Shiraki, Tatsuya Shindo, Nobutaka Imamura,
Toshiyuki Shimizu, and Hiroaki Ishihata
Parallel Computing Research Center, Fujitsu Laboratories Ltd.
E-mail: woods@flab.fujitsu.co.jp

Abstract

Distributed-memory highly parallel computer AP1000+ was developed to support communication functions which were required in parallelizing compilers, such as HPF, and VPP Fortran. The AP1000+ has communication mechanisms which include non-blocking command issue, bulk data transfer, stride data transfer, global reduction and barrier synchronization. In this paper, we simulated effects of queue overflow, related to non-blocking command issue, and stride data transfer using scientific applications.

1 はじめに

分散メモリ型並列計算機はそのスケーラビリティから，大規模問題を解くための次世代スーパーコンピュータの有力候補の一つである．このような並列計算機が広く使われるためには，プログラマにとって不連続なアドレス空間やメッセージ通信等，ハードウェアの詳細を意識せずにプログラミングが行なえることが重要である．特にプログラマに対してグローバルアドレス空間をサポートすることは，プログラミングの容易化や既存ソフトウェア資産の継承という観点から有用である．こうした要求に応えるためにHPF [1]，Fortran D [2]，VPP Fortran [3]などの言語が提案されている．

しかしこれまでの分散メモリ型並列計算機は，必ずしもこれらの言語が必要とする機能を十分にサポートしてはいなかった．HPF 等の言語を分散メモリ型の並列計算機の上で実用化するためには，これらの言語の必要とするデータ通信の機能を考慮したアーキテクチャが求められる．

我々は HPF [4] と VPP Fortran [5] を AP1000 上に実現した経験と他の並列言語の研究から，分散メモリ型並列計算機上にこれらの言語処理系を実現するために必要となる機能を検討し，それらをアーキテクチャでサポートする分散メモリ型高並列計算機 AP1000+ を開発した[6, 7, 8]．

AP1000+ は PUT/GET を通信の基本とし，低オーバーヘッド高スループットの通信を実現すると共に，分散メモリ型並列計算機上に HPF や VPP Fortran 等の言語処理系を実現する際に必要となる，通信終了判定，ストライドデータ転送，バリア同期，グローバル演算などの機能を備えている [9]．

本論文では低オーバーヘッドを実現するためのノンブロッキングコマンド発行に伴うキュー溢れの性能への影響とストライド転送の効果について，NAS パラレルベンチマーク [10] 等の科学技術計算のプログラムを用いて，シミュレーションによって評価する．

2 ノンブロッキング通信

ノンブロッキング通信は各送信ライブラリで，送信完了を待たずに次の処理へ進む方式であり，通信のオーバーヘッドを削減するために重要である．一般には送信するデータを保護するために，送信データを一旦システムの用意した送信バッファにコピーするか，送信の完了を待つ方式などを取る必要がある．これは，送信領域への書き込み等によって送信データが破壊される可能性があるからである．

しかしメッセージ通信を用いる多くのプログラムでは，更新直後のデータを送信するため，送信関数発行後すぐに送信データ領域に書き込み等を行なうことは少ない．このためノンブロッキング送信を用いることで，多くのアプリケーションの実行性能の改善が可能となる [11]．

特に並列化コンパイラでは，データの一括転送によって通信と演算のオーバーラップを図るので，ノンブロッキング通信は不可欠となる．

VPP Fortran では，プログラムをチューニングするために複数のデータの一括転送を指定するディレクティブを用意している．分散メモリ型の並列計算機において，データ転送の処理時間は転送の回数に大きく依存するために，一括転送によって通信オーバーヘッドを削減することが重要であると同時に，通信と演算のオーバーラップによって最適化を行なうためである．

一括転送のためのディレクティブには，SPREAD MOVE と OVERLAP FIX がある．SPREAD MOVE は，配列から配列への複数要素の転送を一括化するための記述である．DOループで書かれた配列間代入に対して，SPREAD MOVE のディレクティブを挿入することでコンパイラに指示する．

OVERLAP FIX は，配列をプロセッサにブロック分散した際に，隣合うプロセッサで境界部分のデータを参照しあう場合のチューニングに用いる．まず互いに参照する境界部分のコピーをそれぞれのプロセッサに持たせ，このコピー部分を最新の値に更新する指示が OVERLAP FIX である．

SPREAD MOVE を用いた一括データ転送の例をList1に示す．List1の例では，グローバル配列Bのデータのうち必要なものをローカルな配列AにSPREAD MOVE で一括転送している．

List 1 SPREAD MOVE の例

```
1   !XOCL SPREAD MOVE
2             DO 200 J=1,M
3               A(J)=B(J,K)
4     200     CONTINUE
5   !XOCL END SPREAD (X)
6   !XOCL MOVEWAIT (X)
```

このSPREAD MOVEによるデータ転送は6行目のMOVEWAITディレクティブまでに完了することになっている．このMOVEWAIT をできるだけ後ろに移動させることで，通信と演算をオーバーラップさせ，プログラムの最適化を図ることができる．

2.1 キューオーバーフローの制御

このようにノンブロッキングで送信を行なうためには，送信コマンドを蓄えるキューが必要になる．通信処理のレイテンシを抑えるためには，ハードウェアでキューを作ることが重要だが，それには大きさに限度がある．そこでハードウェアのキューをキャッシュのように利用して，溢れた場合には主記憶上のソフトウェアによるバッファに書き込むような処理を行なう機構が必要になる．

MSC+ の RAM 内には 5種類のキューがある．送信用のキューには3種類あり，それぞれユーザの PUT，システムの PUT，リモートアクセス用である．PUT がユーザ用とシステム用に分かれているのは，システム・モードになった時にユーザのリクエストを退避せず，システムの PUT を実行できるようにするためである．またリモートアクセス要求のキューが別になっているのはリモートアクセスを PUT より優先するためである．

一方リプライキューは2種類あり，それぞれ，GET のリプライとリモートアクセスのリプライ用である．リプライ

もリモートアクセスを優先している．

各キューの大きさは最大 64 ワードで，AP1000+ では通常の PUT/GET のコマンドパラメータは 8 ワードで構成されるので，キューには最大で 8 エントリまで入れることができる．

しかしノンブロッキングで通信を行なう場合には，8個以上 PUT/GET を利用する可能性がある．このような場合には，MSC+ 内のキューが溢れてしまうので，主記憶上のバッファに自動的に書き込む機構をMSC+ は備えている (図 1)．

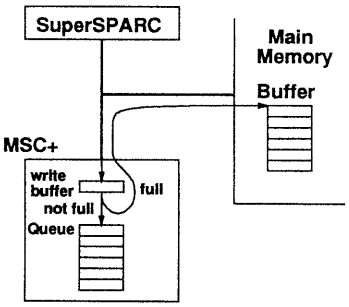

図 1: キューオーバーフローの扱い

キューの溢れの処理は以下のように行なう．MSC+ には1ワード分のライトバッファがあって，SuperSPARC からのキューへの書き込みは必ず成功する．ここでライトバッファに書き込まれたデータはキューに空きがあれば，そのままキューに書き込まれる．もしキューが一杯の時は，主記憶上に予め OS が確保したバッファにデータを書き込む．主記憶上のバッファは一つで，MSC+ 内の5種類のキューから溢れたデータは混ざって書き込まれる．

一旦キューが溢れた後はプロセッサからのキューへの書き込みは全てライトバッファを経由して主記憶上のバッファへ書き込まれる．MSC+ 内の処理が進み，キュー内の最後のコマンドが認識され，DMA が起動されると割り込みが発生し，OS によって主記憶上のバッファから空になったMSC+ 内のキューに入るべきコマンドを選んでキューへデータがロードされる．OS による処理はキューの最後のコマンドによる DMA が起動された直後に開始される．このため，メッセージサイズが大きければ DMA によってデータが転送されている間にコマンドのロードが可能となり，OS による処理のオーバーヘッドをネットワークから隠蔽することができる．

2.2 シミュレーションモデル

メッセージレベルシミュレータ (MLSim) はアーキテクチャの評価を目的としたシミュレーション・ツールであり，メッセージ・パッシングで記述されたプログラムをシミュレートする [11]．このメッセージレベルシミュレータを AP1000 で割り込みを利用したPUT/GET インターフェース [12] に対応するように改良し，AP1000+の性能予測を行なった [9]．

シミュレーションはプロセッサの性能を SuperSPARC が SPARC の8倍，その他の通信パラメータは AP1000+ のハードウェアの仕様から適切に決定して行なった．これ以降このモデルを AP+ モデルと呼ぶ．

図 2 にシミュレーションにおけるキュー溢れの処理モデルを示す．MLSim ではローカルの送信キューの溢れのモデルにだけ対応している．

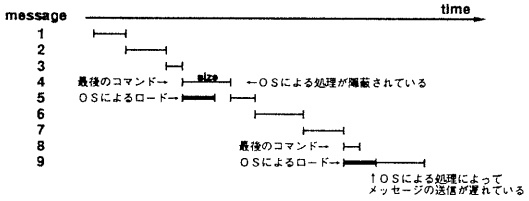

図 2: キュー溢れモデル

MSC+ ではキューの最後のコマンドを認識した時点で割り込みを発生するので，最後のコマンドが指定するメッセージのサイズが大きければ，図2 のメッセージ4と5の間のように OS による主記憶上のバッファからのコマンドのロードの処理をネットワークから隠蔽することができる．

この場合には，シミュレータでは，プロセッサの演算時間をブロックするだけで，メッセージの発信時刻は通常の場合と変わりがないように扱っている．一方，図 2のメッセージ8と9の間の例のように最後のコマンドによるデータのサイズが小さい場合には OS の処理を隠蔽することができない．このような場合には，シミュレータではメッセージの発信時刻を遅らせている．また OS による割り込み処理の時間は通信ライブラリのオーバーヘッドに加算している．

2.3 評価に用いたアプリケーション

評価に利用したのは表1に示す VPP Fortran で書かれたアプリケーションで，NAS パラレルベンチマーク [13] から SP，CG，FTの3つと SPEC ベンチマーク [14] から TOMCATV の計4つである[10]．

表2に各アプリケーションで使われる通信の種類と平均の利用回数を示す．

表 1: アプリケーション

SP	scalar pentadiagonal equations の解を求める．64 × 64 × 64の配列に対し400回の反復を行なう．メモリの制限から，最初の10回をトレースした．
CG	共役勾配法による連立方程式の解法．行列サイズは 1400 で，非零要素数は78184個．
FT	3次元FFT．配列のサイズは 64 × 64 × 64．
TOMCATV	メッシュ生成プログラム．

表 2: 各アプリケーションの通信回数

Application	PE	Send per PE	Gop per PE	V Gop per PE	Sync per PE	PUT per PE	PUTS per PE	GET per PE	GETS per PE	Size of Msg.
CG	16	365.6	810.0	390.0	3135.0	390.0	0.0	0.0	0.0	700.0
FT	16	0.0	12.0	0.0	41.0	0.0	5376.0	5808.0	512.0	867.4
SP	64	1.0	0.0	1.0	42.0	10880.0	0.0	10710.0	0.0	1355.3
TOMCATV	16	0.0	20.0	0.0	80.0	0.0	37.5	37.5	0.0	2056.0

Send:	point-to-point send message	PUTS:	PUT with stride data transfer
Gop:	global operation for a scaler	GET:	point-to-point GET message
V Gop:	global operation for a vector	GETS:	GET with stride data transfer
Sync:	barrier synchronization	Size of Msg:	average message length for each
PUT:	point-to-point PUT message		PUT/GET (bytes) without GET for acknowledge

2.4 キューに溜るコマンド

送信キューが無限に長いとした場合に，送信キューにPUT や GET のリクエストコマンドが何個溜るかをシミュレーションによって求めた．表 3 に AP+ モデルについて，各セルで各アプリケーションを実行中に溜ったコマンド数の最大値の平均を示す．ここでキューの長さとは各送信コマンド(PUT/GET 等)が発行された時点で，いくつ残っていたかを表す．

キューに溜るコマンドの数はアプリケーションによってかなり性質が異なり，非常に長くなるもの(FTとSP)とほとんど溜らないもの(CGとTOMCATV)に分かれる．

この原因を各アプリケーションについて考えてみる．

CG 表2に示すように `send()` が多く用いられている．`send()` は送信領域の保護を保証する通信方式で，送信完了まで通信ライブラリから出て行かない．またグローバル演算やバリア同期がたくさん用いられており，`put()` が連続して発行されていない．このためコマンドがキューに溜らない．

FT `SPREAD MOVE` による一括データ転送が頻繁に起こり，多くの `put_stride()` および `get_stride()` が用いられている．このためキューにコマンドが多く溜る．

SP FT 同様に `SPREAD MOVE` による一括データ転送が頻繁に起こり，多くの`put()` が用いられている．このためキューにコマンドが多く溜る．

TOMCATV `OVERLAP FIX` による一括データ転送が利用されているが，グローバル演算やバリア同期が頻繁に用いられるため，プロセッサによるコマンドの発行が連続して行なわれない．このためキューにはあまりコマンドが溜らない．

2.5 ノンブロッキング通信の効果

ノンブロッキング通信の効果を評価するために，キューにコマンドが溜り易いFT に対し，以下の4つのモデルについてシミュレーションを行なった．

表 3: 送信キューの平均値

Application	AP+
CG	0.0
FT	175.1
SP	374.5
TOMCATV	6.5

モデル 1 キューが存在せず，全ての PUT/GET を送信完了まで，通信ライブラリでブロックする場合．

モデル 2 MSC+ 内のキューに収まる範囲(ここでは8コマンド)で，通信ライブラリでブロックする場合．

モデル 3 全てノンブロッキングで送信し，MSC+ 内のキューに入らないコマンドは主記憶上のバッファに溢れさせて，OS で詰め直す場合．シミュレーションでは，OS による処理の時間を $20\mu s$ と仮定した．このモデルは AP1000+ に対応している．

モデル 4 全てノンブロッキングで送信し，MSC+ 内のキューが無限に長いとした理想的な場合．

図3に各アプリケーションでの性能を4番目のモデルの実行時間を基準にして表す．

ここで区分の意味はそれぞれ以下の通りである．

[Execution time] プログラムが計算を行なう時間で，以下の3つの時間を全体の時間から除いたもの．

[Run time system] VPP Fortran ランタイムシステムが通信を行なうために送り先のアドレスの計算や，ストライドの検出及びストライドパラメータの計算等に要する時間で，通信は含まない．

[Overhead] 通信ライブラリ内での処理時間からアイドル時間を除いたもので，この間プロセッサの演算が妨げられる．モデル 3で OS によるキュー溢れの処理時間はここに含まれる．

[Idle time] メッセージ待ち及び同期成立待ち時間，さらにモデル 1，2でブロックされる時間を含む．

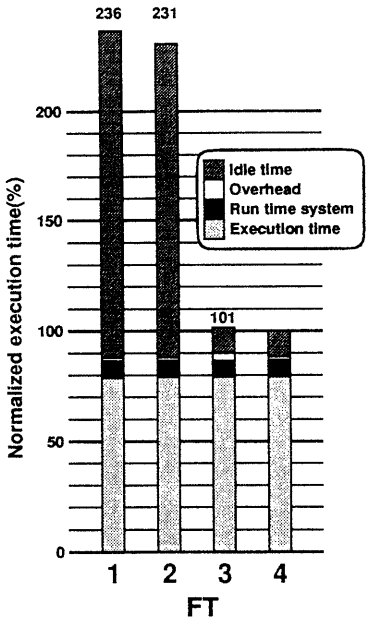

図 3: ノンブロッキング通信の効果

2.6 シミュレーションの評価

モデル 1 では一つ一つコマンドをブロックするので，ブロックによるアイドル時間が大きく，実行時間が理想的な場合(モデル 5)と比べて約2.4倍になっている．

モデル 2 ではオーバーヘッドはモデル1と同じだが，アイドル時間が短くなっており，実行時間は理想的な場合と比べて約2.3倍になっている．しかし，キューの数を1から8に増やした効果はほとんどない．

MSC+ 内のキュー を利用する場合(モデル 3)はキューをコマンドのキャッシュのように利用することに相当し，理想的な場合と比べて約1% 遅くなっている．FT の場合，プログラム実行のほぼ最初から最後までキューには8個以上のコマンドが溜っており，OS による割り込み処理はほぼコマンド8個に一回の割合いで発生している．しかし，それでも全体の性能劣化は約1%になっており，キュー溢れの処理の影響はほとんどないということができる．

モデル 4は無限に長いキューのキャッシュがある場合に相当し，オーバーヘッドとアイドル時間が最も短くなる．

ブロッキングを行なうモデル 1，2 に対して，ノンブロッキングのモデル 3，4 は2倍以上の性能を示している．このようにノンブロッキングによって通信を行なうことはプログラムの実行効率を上げるうえで極めて重要であることが分かった．

このシミュレーションではリモートのリプライキューは無限に長いと仮定している．この仮定を行なった理由は，リプライキューが使われる頻度が送信キューと比べてかなり少なくなる可能性があるため，その溢れの影響は少ないと考えられるからである．

リプライキューを利用するのは GET であるが，表2 で示されている GET の通信は全て PUT のためのアクノレッジに利用されているものである．これは AP+ では，PUT に対するアクノレッジは PUT に引き続いて発行されるGET で代用されるからである．現状の VPP Fortran の実装では全ての PUT に対してアクノレッジが要求されているため，GET の回数が多くなっている．しかしこれは大幅に削減することが可能で，現在このための作業を行なっている．

3 ストライドデータ転送

ストライドデータ転送を利用しているアプリケーション(FTとTOMCATV)について，ストライド転送の効果を調べた [15]．表4 は通信回数がどのように変化するかを示す．FT と TOMCATV についてそれぞれ，上がストライド転送を用いた場合，下がストライド転送を用いなかった場合である．表4に示していない他の通信に関しては表2と同じである．

FT の場合では，通信回数が約54倍になり，メッセージの平均サイズは約48分の1になっている．TOMCATV の場合では，通信回数が257倍になり，メッセージの平均サイズは257分の1になっている．ストライドを使わないと非常に多くの PUT/GET が使われることが分かる．またメッセージサイズも非常に小さくなる．

表 4: ストライド転送による通信回数の変化

	PUT per PE	PUTS per PE	GET per PE	GETS per PE	Size of Msg.
FT ST	0.0	5376.0	5808.0	512.0	867.4
FT	290304.0	0.0	305696.0	0.0	17.8
TC ST	0.0	37.5	37.5	0.0	2056.0
TC	9637.5	0.0	9637.5	0.0	8.0

TOMCATV に関してはメッセージレベルシミュレータによって AP+ モデルでストライドデータ転送の効果を調べた[1]．

図4 では，左側がストライドデータ転送を用いた場合，右側が用いなかった場合で，ストライド転送を用いた場合の全体の実行時間を 100 としている．図4 に示すようにストライドデータ転送を利用しない場合には，実行時間が約1.5倍になっている．ストライドデータ転送を利用するかどうかで 50%もの性能差があり，ストライドデータ転送をサポートすることが重要であることが分かる．

4 おわりに

並列化コンパイラに必要とされる通信機能について，これをサポートするAP1000+ の性能をメッセージレベルシミュレータを用いて，NAS パラレルベンチマークなどの

[1]FT については通信回数が多く，トレースバッファのサイズの制限から，シミュレーションを行なうことができなかった．

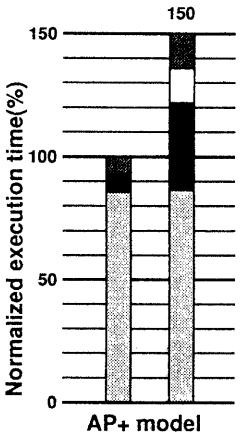

図 4: TOMCATV におけるストライド転送の効果

大規模科学計算のアプリケーションに対する性能を評価した.

シミュレーションによれば FT の場合で, ノンブロッキング通信による効果は大きく, ブロッキングする場合と比べて2倍以上性能が良い. また, ノンブロッキングのためのキュー溢れの OS による処理は, 実行性能を数%しか劣化させないことを示した.

大規模数値計算のプログラムではストライド転送はよく用いられる転送パターンであり, 性能向上に大きく寄与する. ストライドデータ転送をハードウェアでサポートする意義は大きい.

このように並列化コンパイラが必要とする通信機構をアーキテクチャによってサポートする高並列計算機 AP1000+ によって, コンパイラが生成するコードを効率的に実行することができる.

謝辞

日頃御指導, 御助言いただく, 並列処理研究センター石井センター長, 白石担当部長, 池坂主任研究員, ならびに研究室の同僚諸氏, 特に NAS パラレルベンチマークの VPP Fortran による並列化を行なった金城ショーン研究員に感謝いたします.

参考文献

[1] High Performance Fortran Forum. *High Performance Fortran Language Specification Version 1.0*, May 1993.

[2] S. Hiranandani, K. Kennedy, and C. Tseng. Compiler optimizations for Fortran D on MIMD distributed-memory machines. In *Supercomputing '91*, pp. 86–100, 1991.

[3] K. Miura, M. Takamura, Y. Sakamoto, and S. Okada. Overview of the Fujitsu VPP500 Supercomputer. In *COMPCON 93*, pp. 128–130, Feb. 1993.

[4] 萩原純一, 金城ショーン, 土肥実久, 岩下英俊, 進藤達也. HPF コンパイラの実装と AP1000 を用いた評価. SWoPP 琉球 '94 HPC 研究会, Jul. 1994.

[5] 進藤達也, 岩下英俊, 土肥実久, 萩原純一. AP1000 を対象とした VPP Fortran 処理系の実現と評価. SWoPP 鞆の浦 '93 HPC 研究会, Vol. 93-HPC-48-2, pp. 9–16, 1993.

[6] 石畑宏明, 堀江健志, 清水俊幸, 林憲一, 小柳洋一, 今村信貴, 白木長武. AP1000+: デザインコンセプト. SWoPP 琉球 '94 ARC 研究会 (20), 1994.

[7] 小柳洋一, 白木長武, 今村信貴, 林憲一, 清水俊幸, 堀江健志, 石畑宏明. AP1000+: メッセージハンドリング機構(I) – ユーザーレベルインターフェース –. SWoPP 琉球 '94 ARC 研究会 (21), 1994.

[8] 白木長武, 小柳洋一, 今村信貴, 林憲一, 清水俊幸, 堀江健志, 石畑宏明. AP1000+: メッセージハンドリング機構(II) – システムレベルインターフェース –. SWoPP 琉球 '94 ARC 研究会 (22), 1994.

[9] 林憲一, 土肥実久, 堀江健志, 小柳洋一, 白木長武, 今村信貴, 清水俊幸, 石畑宏明, 進藤達也. PUT/GET インターフェースのハードウェアサポートによる並列プログラムの効率的実行. 並列処理シンポジウム JSPP '94, pp. 233–240. 情報処理学会, 1994.

[10] 金城ショーン, 進藤達也. VPP Fortran を用いた NAS Parallel Benchmark の並列化と AP1000 を用いた評価. SWoPP 琉球 '94 HPC 研究会, Jul. 1994.

[11] T. Horie, K. Hayashi, T. Shimizu, and H. Ishihata. Improving AP1000 parallel computer performance with message communication. In *The 20th Annual International Symposium on Computer Architecture*, pp. 314–325, May 1993.

[12] 林憲一, 堀江健志. アクティブ・メッセージによる並列プログラム実行性能の改善. SWoPP 鞆の浦 '93 プログラミング研究会, Vol. 93-PRG-13-17, pp. 129–136, 1993.

[13] D. Bailey, J. Barton, T. Lasinski, and H. Simon. The NAS Parallel Benchmark. Technical Report RNR-91-002 Revision 2, NASA Ames Research Center, Moffett Field. CA 94035, August 1991.

[14] Waterside Associates. *The SPEC Benchmark Report.* Fremont, CA, Jan. 1990.

[15] 土肥実久, 林憲一, 進藤達也. ストライドデータ転送機構を用いたコード生成. SWoPP 琉球 '94 HPC 研究会, Jul. 1994.